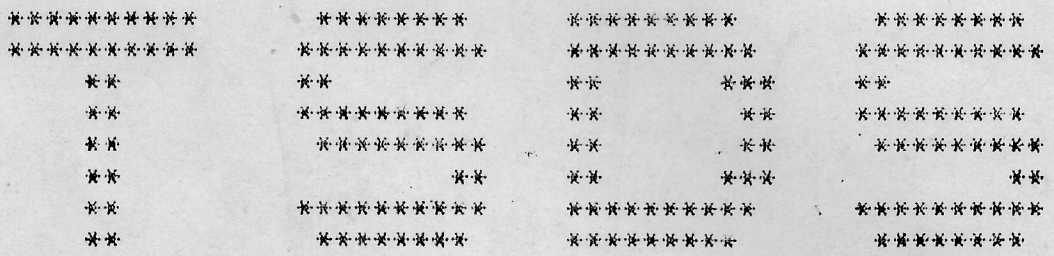

Total Software Development System
(Sistema de desarrollo total del software)
Versión commodore 64/128

Copyright 1985 por NoSync y Kevin P. Pickell
#103 2334 Marpole St
Port Coquitlam, BC V3C 2A2

Escrito por Kevin P. Pickell
en Junio de 1986

Traducción y revisión del texto por:
Mario Daniel Villarreal

El SYSEDT, es uno de los componentes del "TOTAL SOFTWARE DEVELOPMENT SYSTEM (TSDS)". Para cargar al TSDS, inserte su disco TSDS en el drive y si está en modo 64 o en una C=64 DESCONECTE SU FASTLOAD y tipee: LOAD"0:*",8,1 (RETURN). El SYSEDT se autoarrancará. En el caso de estar con una C=128 en modo 128, inserte el disco TSDS y resetee la computadora. El programa se autoarrancará solo.

SYSEDT es un sistema operativo que trabaja en conjunción con el BASIC, mejorando notablemente el editor, especialmente en la C=64. El SYSEDT, implementa comandos como: directorio, repetición de teclas (40 columnas), scroll, comandos de disco, y comandos de impresora. Los comandos de disco se preceden por cualquiera de los siguientes signos: ">" o "!" o "*" o por "@". Ellos son:

COMANDOS EXTENDIDOS DEL SISTEMA OPERATIVO DEL DISCO

```
>n  Formatear diskette
>i  Reconocer diskette
>v  Validar diskette
>d  Copiar diskette (sólo para doble drive)
>c  Copiar filas
>r  Renombrar filas
>s  Borrar filas
>$  Directorio
>l  Listar filas
```

(Recuerde que SIEMPRE cualquier comando de disco puede iniciarse indiferentemente con ">", "*", "@", o "!").

```
Fomatear:   @NO:nombre,identificador
Renombrar:  @RO:nombre nuevo=O:nombre viejo
Copiar:     @CO:nombre nuevo=O:nombre viejo
```

COMANDOS EXTENDIDOS DE EDICION

#	Ver estado del editor, y versión del mismo
/	Cargar un programa en área BASIC
↑	Cargar y correr un programa en área BASIC
BLOAD	Cargar un programa en su real dirección
BRUN	Cargar programa en L. M. y saltar a dirección de carga
APPEND	Apendizar una fila o programa BASIC al en memoria
ASM, ASSEMBLE	Ensamblar un archivo fuente
AUTO	Activar numerador automático de líneas
CDISK	Cambiar número de periférico del drive
CHANGE	Cambiar un patrón de caracteres por otro
COLOUR	Poner color de borde y fondo
CONV	Conversor a binario, decimal, hexadecimal y ascii
CPUT	Salvar archivo en disco compactando espacios
DELETE	Borrar un rango de líneas
DFREE	Ver nombre del diskette y espacio libre en el mismo
DISK	Seleccionar drive
EDIT [S] [B]	Seleccionar editor BASIC o editor de archivos
EXEC	Ejecutar un archivo con comandos extendidos de edición
FIND	Hallar la ocurrencia de un patrón de caracteres
FVER	Verificar actual archivo en memoria con el del diskette
GET	Cargar un archivo en memoria
HELP	Ver los comandos extendidos existentes
KEY	Definir una tecla de función
KEYS [-]	Activar/desactivar teclas de función
KEYC	Borrar todas las teclas de función
KEYL	Listar todas las teclas de función definidas
KEYR	Fijar velocidad de repetición del teclado (40 columnas)
KILL	Desactivar definitivamente el editor
LOADAT	Cargar un programa en dirección indicada
LIST	Listar el archivo o programa en memoria
LOG	Mostrar inicio y fin de carga de un programa
OLD	Recuperar un programa borrado de la memoria
PASSWORD	Palabra de pase protectora de archivos
PRINTER	Elije periférico de la impresora y espacio interlineal
PUT	Graba un archivo sin compactar espacios
RENUMBER	Renumerar todo o parte del texto en memoria
RESTORE	Restaurar editor a estado inicial
FAST	Activar sistema a modo rápido (por defecto)
SLOW	Pasar sistema a modo lento

ACCION DE LOS COMANDOS EXTENDIDOS DE DISCO

Dado que el drive tiene su propio sistema operativo, los comandos de disco son enviados y la computadora queda liberada. El Sistema Operativo del Disco ('DOS') recibe desde el editor cualquier comando ejecutable, sin necesidad de apertura de canales especiales (de ello se encarga el editor), con la condición de que el primer caracter sea uno de los antedichos (>,!, etc.). Para enviar cadenas con caracteres ascii no imprimibles, pueden formarse variables o cadenas entre comillas, o funciones alfanuméricas BASIC -por ej: chr$()-, concatenados por el signo "+". El caracter nulo, se envía así: "". Verbigracia: >""+A$+CHR$(1)
Para leer el canal de mensajes del drive, basta con un caracter de comando DOS: (>,!, etc)

ACCION DE LOS COMANDOS EXTENDIDOS DE IMPRESORA

Precediendo cualquier comando SYSEDT o BASIC con un asterisco (*), se desvía la salida al periférico de impresora seleccionado. Recuerde usar el comando PRINTER para seleccionarlo (4 por defecto).

Comando: >L
Función: Listar un programa o archivo del disco
Sintaxis: >L "nombre"
Descripción: La fila se lista como un programa BASIC (si lo es), o como una fila fuente, si es un archivo secuencial. El contenido de la fila se lista por pantalla o impresora. No se carga, por lo que no altera el contenido de la memoria. El listado puede pausarse con la barra espaciadora y continuarse presionando una tecla cualquiera (excepto STOP). La impresión puede abortarse con la tecla stop. El comando *>L"nombre" envía el listado por impresora.

Comando: >S
Función: Borrar fila/s.
Sintaxis: >S[número de drive]:"nombre"
Descripción: Una fila o grupo de filas puede ser borrado tipeando ">S0:", en el márgen derecho de un listado de directorio. Todos los "wildcards" son aceptados.

Ejemplos: >S0:"FI*"
>S1:A?1

Comando: >$
Función: Listar el directorio del disco
Sintaxis: >$[0:"patrón"]
Descripción: El comando ">$" muestra el directorio del mismo que si hubiera sido cargado y listado de la memoria, con la diferecia de que no se carga, y no altera la memoria. Todos los patrones del DOS se permiten. El comando "*>$" envía el directorio por impresora. También puede pausarse con la barra y cortarse mediante STOP.

COMANDOS EXTENDIDOS DEL EDITOR

Comando:#
Función: Muestra el estado del editor y su actual versión.
Descripción: El muestreo se realiza en el orden siguiente: Editor:Automático:Periférico del drive:Teclas de función:Periférico del drive. Como todo esto sale en inglés, se verá así:
#
** SYSEDT V1.x JANUARY 1986 (TSDS) **
(c) 1985 BY NOSYNC and KEVIN P. PICKELL
EDIT:x:AUTO:x:DISK:x:KEY?:x:PRNT:x

La "x" tras V1., será el número de versión. La "x" tras EDIT, puede ser una "S", indicando que se está en modo de editor fuente (para archivos secuenciales). Podría ser una B, indicando que se está en editor BASIC. La "x" tras AUTO será una "Y" (de Yes) si está activo el automático, en cuyo caso además se verá el incremento, o una N (de No) en caso contrario. La "x" tras DISK, puede ser 8 ó 9, según la asignación hecha al drive (8 por defecto). La "x" tras KEY, puede ser una "Y" si están activas las teclas de función, o una N de lo contrario. Finalmente, la "x" tras PRNT, puede ser un 4 o un 5, según la asignación hecha a la impreso-

ra (4 por defecto). A través de los distintos comandos, el número de periférico puede ser variado en el rango 0 a 31, pero se debe respetar el número establecido en el sistema operativo KERNAL para cada periférico.

Comandos: /, ↑, BLOAD, BRUN
Función: Cargar /, BLOAD
Cargar y correr ↑, BRUN
Sintaxis: comando "nombre"
Descripción: Estos comandos son rápidos y sencillos. Permiten cargar y correr programas del disco. Si se usan sobre el listado de un directorio, todos los números antes de una comilla son ignorados. El comando "/" carga el programa en el área BASIC. La "↑", lo mismo, pero además lo corre tras cargarlo (run). BLOAD es similar a "/", pero el programa se carga en la dirección grabada en el disco. BRUN, además salta a dicha dirección. Si a "↑" no le sigue un nombre, ejecuta el programa en memoria.

Comando: APPEND
Función: Apendizar una fuente del disco o programa BASIC al memorizado.
Sintaxis: APPEND "nombre"
Descripción: La función APPEND toma un archivo fuente del disco o programa BASIC y lo añade al final del actual en memoria.

Comando: ASSEMBLE, ASM
Función: Ensamblar una fila fuente del disco.
Sintaxis: ASM "fila fuente" TO "programa objeto". (C=64)
Descripción: Este comando hace arrancar al ensamblador SYSASM. Si se requiere un listado, use un "*" precediendo al comando. El listado intenta salir por impresora, pero si está apagada, saldrá por pantalla en formato de 80 columnas. Si se ha dado un nombre de programa objeto, el código de máquina, se almacenará en ese programa. Tras el ensamble, puede tipear "LIST", para ver todos los mensajes de error ocurridos durante el mismo. El comando ASSEMBLE o ASM borra cualquier programa o archivo que estuviese en memoria, antes de su ejecución.

Para una C=128, el comando ASM o ASSEMBLE toma una sintaxis más poderosa:

ASM	Ensambla el código fuente actualmente en memoria. No genera código objeto.
ASM TO	Ensambla el código fuente actualmente en memoria. El código objeto se almacena en la ram 0.
ASM "nombre"	Ensambla la fila fuente indicada. No genera código objeto.
ASM "nombre" TO	Ensambla la fila fuente indicada. El código objeto se almacena en la ram 0.
ASM TO "nombre"	Ensambla el código fuente actualmente en memoria. El código objeto se graba en el disco con el nombre dado.
ASM "nombre 1" TO "nombre 2"	Ensambla la fila fuente dada en "nombre 1" El código objeto se graba en disco con "nombre 2"

Para listar los posibles errores ocurridos previamente se debe cargar el archivo de errores, así: GET "ERRORS". Este archivo se localiza nada menos que en la ram del drive (!). Cabe aconsejar prudencia en esta acción pues su carga borra nuestro código fuente de la memoria definitivamente.
Otro detalle a observar cuidadosamente, es el de NO ensamblar en las siguientes direcciones, pues allí reside nuestro sistema:

```
RAM 0   $0C00 - $1000  ;intérprete de comandos.
RAM 1   $1000 - $57FF  ;Nuestro sistema.
RAM 1   $5800 - $FEFF  ;Tablas de símbolos y Macros.
```

Estas direcciones no deben ser ocupadas por ninguna carga o ensamble ya que provocaría la autodestrucción del sistema.

surgiría una nueva línea, con el número de línea dado + 10.
Para desactivar la auto-numeración de líneas se tipea AUTO y se le da entrada.

Comando: CDISK
Función: Cambiar el número de periférico a la disquetera, mediante software.
Sintaxis: CDISK <actual periférico> TO <nuevo periférico>
Descripción: Si tiene más de una disquetera conectada a la computadora, para diferenciar entre una y otra, se debe asignar un número distinto a cada una.
El proceso es el siguiente: Encienda o deje encendida una sola unidad. Luego tipee CDISK 8 TO 9 y pulse return. Esa unidad -considerando que efectivamente era la 8 (lo que es por defecto si no alteró esto por hardware)- ahora se diferencia de la otra por ser la 9. Ya está en condiciones de encender la unidad apagada.
Al apagarse la disquetera o resetear el sistema, ésta, vuelve a su número de periférico propio.

Comando: CHANGE
Función: Cambiar la ocurrencia de un patrón por otro.
Sintaxis: CHANGE [B][E][V][P][C][R]["]/PATRON ACTUAL/["]/NUEVO PATRON/,[rango]
Descripción: Tras encontrar la cadena de caracteres denominada patrón actual, y sólo si la misma cumple las condiciones dadas en los parámetros, se cambiará el patrón actual por el nuevo. La "/" puede ser cualquier caracter no contenido en ninguna de las dos cadenas.

Parámetros

B	si está al comienzo de línea.
E	si está al final de línea.
V	si es una variable o etiqueta.
P	activar wilcard ("#" como tal).
C	si en área de comandos, no entre comillas, o línea DATA.
R	cambiar el remanente de la línea.
"	activar modo comillas (no permite compactación de línea).

Las opciones B, E, V, P, C y R, pueden combinarse para conseguir la selectividad deseada.

RANGO

El rango tiene los siguientes formatos: línea inicial-línea final
línea inicial-etiqueta final
línea inicial-
-línea final
etiqueta inicial-línea final
etiqueta inicial-etiqueta final
etiqueta inicial-
-etiqueta final

```
Ejemplos: CHANGE/pedro//juan/                ;cambia pedro por juan
          CHANGE BV/loop1//rulo1/,10-        ;cambia loop1 por rulo1 si loop1 está al
                                             ;inicio de línea y es una etiqueta,
                                             ;y está desde la línea 10 en adelante.
```

Comando: COLOUR
Función: Poner color de fondo y borde. (40 columnas)
Sintaxis: COLOUR fondo,borde
Descripción: Como lo indica su función.

Comando: HELP
Función: Muestra todas las palabras reservadas (comandos) del SYSEDT.
Sintaxis: HELP
Descripción: Como lo indica su función

Comando: CONV
Función: Conversión numérica de distintas bases.
Sintaxis: CONV valor
Descripción: El valor especifica en qué base se expresa anteponiendo un prefijo de declaración, según la tabla siguiente:

sufijo	
nada	valor decimal
%	valor binario
@	valor octal
$	valor hexadecimal
" .	valor ascii

Tras dar entrada al valor, se muestra la conversión en el siguiente orden: decimal, hexadecimal (2 bytes), binario (4 nibbles) y ascii (2 caracteres).

Comando: DELETE
Función: Borrar un rango de líneas del programa o fila fuente en memoria.
Sintaxis: DELETE rango
Descripción: Remueve una o más líneas a un tiempo. No puede borrar todo el programa, para ello, se usa el comando NEW. El rango es el descripto en el comando CHANGE.

Comando: DFREE
Función: Visualizar el nombre del diskette y sus bloques libres.
Sintaxis: DFREE
Descripción: Este comando es una vía rápida para ver el espacio libre de un diskette sin tener que ver todo el directorio.

Comando: EDIT
Función: Seleccionar modo de edición (filas fuentes o programas BASIC)
Sintaxis: EDIT [B] [S]
Descripción: El editor SYSEDT puede ser usado para editar programas BASIC o fila fuente. Por defecto, está en modo fuente. El comando EDITS pone al editor en modo fuente y el comando EDITB en modo BASIC.

Comando: EXEC
Función: Ejecutar un archivo de texto con comandos SYSEDT
Sintaxis: EXEC "nombre"
Descripción: EXEC es un comando extremadamente poderoso el cual permite preparar una larga lista de comandos los cuales pueden ser entrados automáticamente, como si hubieran sido entrados desde el teclado. "EXEC" no puede ejecutar archivos de más de 12 kb. El uso común de este comando suele ser: definir teclas, setear número de periférico de la impresora y/o el drive o cualquier cosa que fuera necesario realizar repetidas veces.

Comando: FIND
Función: Encontrar la ocurrencia de un patrón dentro del programa o fuente memorizado.
Sintaxis: FIND [B][E][V][P][C][~]/PATRON/[,rango]
Descripción: Según su función. Los parametros ya están descriptos en el comando: CHANGE.

Comando: FVER
Función: Verificar un archivo fuente en disco con el de en memoria.
Sintaxis: FVER "nombre"
Descripción: El comando FVER trabaja sólamente con archivos fuente comparando el contenido de la memoria, con la fila especificada. Después que ha cotejado ambos y son iguales, se visualiza el mensaje VERIFY OK!. De lo contrario, se visualiza el mensaje ?VERIFY ERROR.

Comando: GET
Función: Cargar una fila fuente del disco.
Sintaxis: GET "nombre"
Descripción: El comando GET carga y va asignando un número consecutivo de línea, empezando en 1000 con un incremento de 10.

Comando: KEY
Función: Define una tecla con una función especial.
Sintaxis: KEY "tecla",expresión.
Descripción: Este comando permite a cualquier tecla ser definida como una cadena de carcteres. Hasta 15 teclas pueden definirse cuya longitud total, -la suma del conjunto-, no debe superar los 255 caracteres. Una tecla definida no tiene efecto si está activo el modo comillas. Para borrar la definición de una tecla cualquiera, se debe tipear KEY "x","", donde x es la tecla a limpiar. Para facilitar la formación de cadenas en modo comillas, RETURN puede incluirse con la combinación SHIFT Z y las comillas con SHIFT Q. La tecla a definir puede determinarse por una expresión, pero debe empezar con comillas. De este modo, puede obtenerse con otros comandos de tecla, por ejemplo, chr$().

Ejemplo: KEY "&",">$"
KEY ""+chr$(131),"load "

Comando: KEYS
Función: Activar/desactivar teclas de función definidas.
Sintaxis: KEYS [-]
Descripción: Por defecto, están activas. Para desactivarlas se tipea "KEYS-", y para reactivarlas "KEYS".

Comando: KEYC
Función: Borrar todas las teclas definidas.
Sintaxis: KEYC
Descripción: Según su función.

Comando: KEYL
Función: Listar todas las teclas definidas y su definición.
Sintaxis: KEYL
Descripción: Según su función.

Comando: KEYR
Función: Fijar la velocidad de repetición del teclado.
Sintaxis: KEYR retardo,repetición
Descripción: El comando KEYR, fija el retardo y repetición inicial en 'jiffies' (1/60 de segundo) para las teclas con autorepetición. Esto, incide en la velocidad del scroll de 40 columnas.

Comando: KILL
Función: Desactivar el SYSEDT
Sintaxis: KILL
Descripción: Desactiva definitivamente al SYSEDT. Para reactivarlo hay que reencender el sistema.

Comando: LIST
Función: Listar el archivo o programa en memoria.
Sintaxis: LIST [rango]
Descripción: Similar al comando LIST del BASIC, con la ventaja de no emitir el mensaje READY ni el BREAK/READY. Esto implica un listado por impresora libre de cadenas espúreas. Mediante las teclas del cursor se puede "scrolar" hacia arriba y hacia abajo el listado en pantalla. El rango es el mismo descrito en el comando CHANGE. LIST sin rango lista toda la memoria. Precedido por "*" el listado se desvia a la impresora.

Comando: LOADAT
Función: Cargar un programa en la dirección especificada.
Sintaxis: LOADAT "nombre",dirección
La dirección puede ser expresada según el formato indicado el comando CONV.
Descripción: El comando LOADAT permite que un programa sea cargado en una dirección especifica. El programa es cargado siempre sobre la RAM, aun cuando, en todo o parte, se direccione sobre el area de entrada salida ($D000 a $DFFF). Luego de la carga, la dirección original y la relocación se visualiza en la pantalla.

Ejemplo: LOADAT"AUTOBOOT",$7000
WAS $0102 TO $024A
IS $7000 TO $7148

Comando: LOG
Función: Mostrar la dirección inicial y final de un programa grabado en disco.
Sintaxis: LOG "nombre"
Descripción: Según lo describe su función.

Ejemplo: LOG "AUTOBOOT"
$0102-$024A

Comando: OLD
Función: Recuperar un programa o texto fuente después de un comando NEW.
Sintaxis: OLD
Descripción: Si ud por accidente borra la memoria con NEW y quiere recuperarla, tipee OLD. La recuperación no será perfecta si ocurren errores de sintaxis entre el NEW y el OLD, esto para la C=64, dado que la C=128 permite hasta resetear la máquina y luego recuperar el programa (!).

Comando: PASSWORD
Función: Proteger archivos fuente.
Sintaxis: PASSWORD
Descripción: El comando permite encriptar archivos fuente como una medida de seguridad. Este es llamado tipeando PASSWORD y dando entrada. Seguidamente, se tipea la palabra de pase -cualquiera que se le ocurra- y se le da entrada. Para la desprotección, vuelva a llamar al comando y de entrada. ¡No olvide su palabra de pase!, pues sin ella no se puede recuperar el archivo. Los archivos EXEC no pueden se encriptados. El programa "CPASS", permite proteger o desproteger un grupo de programas mientras se copia de un disco a otro, o dentro del mismo disco.

Comando: PRINTER
Función: Fijar el número de periférico por defecto de la impresora.
Sintaxis: PRINTER periférico[,L]
Descripción: El comando printer define el número de periférico por defecto de la impresora, el cual será usado por el prefijo-comando *. El comando printer, permite múltiples operaciones con la impresora y con el sufijo L, fija el incremento interlineal (line feed).

Comando: PUT/CPUT
Función: Enviar el programa de texto o archivo fuente al disco, como un archivo secuencial.
Sintaxis: PUT "nombre" ó CPUT "nombre"
Descripción: El comando almacena en disco el archivo fuente presente en memoria. Cada línea es enviada con un retorno de carro al final de la misma. PUT, envía las líneas textualmente. CPUT, compacta las líneas extrayendo todos los espacios no imprecindibles. Antes de grabar, si ya existe un archivo con el mismo nombre, (por ejemplo, una versión anterior) la máquina pregunta si lo reemplaza. Se debe contestar con Y (sí) o N (no).

Comando: RENUMBER
Función: Renumerar todo o parte del programa o archivo fuente en memoria.
Sintaxis: RENUMBER <nuevo número inicial>,<incremento> [,rango]
Descripción: El comando ejecuta su función y si se encuentra en modo BASIC renumera las líneas con IF...THEN, IF...GOTO, GOTO, GOSUB, ON...GOTO, ON...GOSUB, LIST, RUN. Todas las referencias a líneas no existentes quedan con el prefijo ! antecediendo al antiguo número de línea. El rango es el mismo ya descripto en el comando CHANGE.

Ejemplos: RENUMBER 1000,10
RENUMBER 1000,10,500-560
RENUMBER 1000,10,LOOPSTART-LOOPEND

Comando: RESTORE
Función: Retomar todas las condiciones por defecto del SYSEDT.
Sintaxis: RESTORE
Descripción: Según su función.

Comando: FAST
Función: Pasa el drive 1541 (o al 1571 en modo 1541) al sistema DOS rápido, propio del SYSEDT.
Sintaxis: FAST
Descripción: El comando realiza su función, y afecta a los comandos: /, ↑, get, log, bload, brun, loadat. Por defecto esta activo. Cabe acotar que en modo 128, este comando es reemplazado por el FAST del BASIC.

Comando: SLOW
Función: Pasa el drive 1541 (o al drive 1571 en modo 1541) al sistema DOS lento, propio del Commodore 64.
Sintaxis: SLOW
Descripción: El comando realiza su función y puede ser necesario en caso que el drive no esté bien alineado o sucio provocando errores en modo fast. Cabe acotar que en modo 128, este comando es reemplazado por el SLOW del BASIC.

INTRODUCCION AL MACRO-ENSAMBLADOR SYSASM

Un ensamblador es un programa que toma un archivo fuente escrito en formato simbólico y lo convierte en código de máquina, para que el microprocesador pueda ejecutarlo. En lenguaje de máquina, la programación se hace totalmente distinta a otras clases de programación, tales como BASIC, dado que uno tiene que ordenarle al micro exactamente lo que debe hacer, usando instrucciones que manipulan datos en su más simple expresión.
Naturalmente devienen grandes ventajas sabiendo cómo hacerlo. Por decir una, los programas escritos en lenguaje de máquina se ejecutan muchísimo más rápido que los escritos en cualquier otro lenguaje. Además, hay algunas cosas que podríamos querer hacer las cuales son imposibles en BASIC, tal como trabajar con interrupciones. El esfuerzo invertido en aprender código de máquina se recompensa en alto grado, por el nuevo mundo que se nos abre.
El llamado código de máquina, no es más que una secuencia de bytes que el procesador ejecuta. Esto, para un ser humano, no es fácilmente inteligible ni manipulable. Para ayudarnos en esta tarea, se usa un ensamblador, tal como el SYSASM.
El SYSASM nos permite escribir un programa en formato simbólico. Se usan mnemónicos para representar la operación que el procesador puede hacer, y por específicos modos de direccionamiento, usando etiquetas y variables, se crea un código más legible, inteligible y más simple de modificar. El propósito real del ensamblador es alejar lo máximo posible al programa de su verdadero aspecto, sin disminuir en lo más mínimo su habilidad para relacionarse con el hardware de la máquina. Un ensamblador toma un programa abstracto denominado código fuente y lo convierte en un programa de lenguaje de máquina llamado código objeto. Por ejemplo, una línea en el formato LDA #10 (fuente) pasa a este 1001010010000101O (objeto). El ensamble es llevado a cabo por el comando ASM del SYSEDT.

UN ENSAMBLE

Un ensamble es un proceso realmente complicado. Pero la cosa más importante que debemos saber acerca del ensamblador SYSASM es que se trata de un ensamblador de dos pasos. Esto significa que el SYSASM lee el código o archivo o texto fuente dos veces. En el primer paso, determina cuántos bytes de memoria ocupa cada sentencia o línea fuente. Esto, permite conocer la locación o dirección de cada una de ellas y posicionar correctamente el PC (contador de programa) del micro. A todas las etiquetas se les asigna su valor en el primer paso. Ciertos directivos -detallados más adelante- operan en el primer paso. Cuando el ensamblador lee la sentencia del archivo fuente, se realiza el siguiente proceso: Primero, analiza (separando secciones) la sentencia en su principales componentes. Hecho esto, ha determinado que clase de código de operación o de directivo contiene la línea.

En el primer paso se asignan los símbolos o etiquetas. En el segundo paso, los directivos son ejecutados por completo y las instrucciones decodificadas. Las expresiones son evaluadas y el código objeto se genera. Finalmente se da un listado si es requerido. Si hay algún error, se lista la línea de su ocurrencia bajo cualquier circunstancia y el código objeto es generado con un cero en su primer byte (BRK). Tras todo esto, el ensamblador SYSASM va en busca de la próxima instrucción.
Mientras el SYSASM está ensamblando, se puede presionar la tecla RUN/STOP para interrumpir temporalmente el ensamble. Tras presionar la tecla se visualiza el mensaje: Q-QUIT, C-CONTINUE. De modo que pulsado la tecla Q se aborta el ensamble, y con la C se lo continúa.
En un ensamble largo, el SYSASM va dando una idea de lo que está haciendo imprimiendo cualquier cambio de archivo bajo cualquier circunstancia. También, durante el ensamble, la línea fuente corriente y la posición del contador de programa se visualiza en pantalla.
Uno de los principales problemas con el ensamble de archivos, es que al concluir se visualice un mensaje como: ERRORS=0025 y ya haber perdido de vista en la pantalla (por el scroll) la línea donde ocurrió el error. SYSASM resuelve este problema. Cuando ocurre un error el mensaje y la línea listada se salva en memoria. Tras el ensamble, se tipea LIST (o *LIST para impresora) para imprimir todos los mensajes de error. Si hay más mensajes de los que entran en 8Kb los mensajes excedentes no son salvados. Cabe acotar que para la C=128, se puede cargar el archivo "ERRORS", según se describe en el comando ASM del SYSEDT. Diríjase al mismo para más información.
Para ver cómo trabaja el SYSASM pruebe el siguiente programa ejemplo: (para modo C=64). Primero, pase su C=128 a modo C=64 -si tiene una- y cargue el SYSEDT. Entre el comando NEW para borrar cualquier programa basic previo posible. A continuación, tipee el programa que sigue:

```
10 ;*** PROGRAMA EJEMPLO ***
20 *=$1000
30 ;
40 LDX #0
50 LOOP TXA
60 STA $0400,X ;MEMORIA DE PANTALLA
70 LDA #1      ;COLOR BLANCO
80 STA $D800,X ;MEMORIA DE COLOR
90 INX
100 BNE LOOP
110 RTS
120 .END
```

El programa precendente llena una porción de la parte superior de la pantalla con todos los códigos de pantalla existentes. Como se ve en el listado en la línea 40, el registro X se carga con el valor cero. Luego el contenido del registro X se transfiere al acumulador y se almacena en pantalla en posiciones indexadas por X. Luego el color blanco (código 1) es depositado en el area de color, usando el mismo modo indexado. Después, el registro X es incrementado (generando un nuevo desplazamiento) y testeado sobre si a vuelto a cero. En caso negativo, sigue el loop y en caso afirmativo se sale de la subrutina mediante RTS.

Destaquemos algunos elementos del listado. No hay marcas especiales que podrían verse en un listado normal de ensamblador. La sentencia en línea 50 la cual tiene una etiqueta llamada LOOP, no se diferencia por ningún caracter especial del resto de los componentes de la línea. En segundo lugar, la sentencia de la línea 120 comienza con un punto, y no se trata de ningún mnemónico de ensamblador.
¿Qué hacen todos estos elementos?
Los archivos assembler, son de formato libre; ninguna marca o caracter especial se requiere para distinguir los distintos elementos. El ensamblador determina si la línea es una mnemónico de lenguaje máquina, una sentencia con una etiqueta, o uno de sus propios directivos. El ensamblador sabe que una sentencia no tiene etiqueta, si la primera palabra en la línea es uno de los 56 mnemónicos del set del micro 6502, 6510, 8502, etc. .Esto significa que no podemos usar una etiqueta que sea absolutamente igual a uno de esos mnemónicos, por ejemplo, NOP o LDA, etc. Si la sentencia comienza con una palabra que no es un mnemónico, se asume que esto es una etiqueta. Las palabras que comienzan con puntos o símbolos especiales están orientados a ser directivos, puesto que una etiqueta debe empezar con una letra.

FORMATO DE UNA LINEA EN ENSAMBLADOR

El ensamblador SYSASM va más allá del formato libre. en SYSASM se puede rellenar las sentencias con espacios para hacer más clara y cómoda la lectura. Se pueden insertar entre operadores, mnemónicos y operandos, tal como lo permite el BASIC. Esto significa que puede haber más de un espacio entre etiqueta y código de operación y entre éste y el operando. También se puede compactar el espacio al mínimo posible; esto ahorra espacio en el disco y acelera el ensamble.
En el listado fuente de nuestro ejemplo anterior, *= es un directivo que fija al contador de programa. También .END, es un directivo que señala el fin del ensamble. Los directivos son tratados al detalle más adelante.
Ahora que se ha explicado un poco más el programa, estamos listos para hacerlo funcionar. Lo primero es tener un disco a disposición para grabar el texto fuente. Para lograrlo nos valemos del comando PUT"ASM.EJEMPLO" dándole return. Hecho esto, tipeamos ASSEMBLE "ASM.EJEMPLO" TO "COM.EJEMPLO" y le damos entrada. Antes de continuar, aclaremos que las siglas "ASM." incluídas en el nombre fuente, no son obligatorias, sino que se trata de una idea práctica que aportamos para distinguir en el directorio del disco los archivos de texto fuente escritos en ensamblador. Así, entrando el comando @$:ASM.*, veremos el directorio de todas las filas fuentes. Igual comentario cabe para las siglas "COM." .Entrando el comando @$:COM. veremos el listado de todos los programas en lenguaje de máquina. Entonces, contaremos con un directorio ordenado y cada texto fuente llevará el mismo nombre que su par en código de máquina, diferenciándose claramente por su declaración de tipo -ASM, o COM-. No obstante, los nombres de fichero fuente y fichero ejecutable quedan liberados a su arbitrio. Hecha la aclaración pertinente, continuamos.
De inmediato, se visualiza el mensaje del assembler para avisarnos que está en acción. Tras unos segundos recuperamos el cursor. De aparecer algun error, retomamos el texto fuente entrando el comando GET"ASM.EJEMPLO" y chequeamos si hemos tipeado correctamente. Ante una equivocación, corregimos y repetimos el proceso de ensamble. Si no tenemos errores, debemos cargar y ejecutar el programa

"COM.EJEMPLO". Podemos hacerlo entrando el comando BLOAD"COM.EJEMPLO". Limpie la pantalla y Ubique el cursor aproximademente en la línea central de la misma y en su esquina izquierda, para no arruinar el efecto del programa, y tipee SYS4096 con return. De inmediato, la parte superior de la pantalla se llenará con todos los códigos imprimibles, en color blanco, y el cursor estará a nuestra disposición. Si algo de esto no ocurre, no se ha seguido fielmente la explicación y se ha cometido algún error.

FORMATO DE UNA LINEA FUENTE

Vamos a entrar en más detalles sobre el formato de una línea en Macro Assembler SYSASM. He aquí sus parámetros:

[ETIQUETA] <espacio> <CODIGOS DE OPERACION> <espacio> <OPERANDO> [;COMENTARIO]

Como notamos previamente la entrada es de formato "libre", de modo que el escribir textos fuentes no tiene complicaciones. Por otro lado, Ud. podría pensar que el listado en formato libre no es claro para la lectura y un poco enjundioso para trabajarlo; que es difícil distinguir las etiquetas, instrucciones y comentarios porque está todo junto; y esto, puede ser tan cierto como falso. Es posible clarificar hasta el exceso una línea fuente, de manera que a voluntad escribamos cada elemento de la línea en una posición, dejando para ello los espacios necesarios entre uno y otro.

ETIQUETAS

Cualquier sentencia normal en SYSASM puede llevar etiqueta. La etiqueta es asignada con el valor del contador de programa, al inicio del ensamble de esa línea. Por lo general, se etiquetan sentencias a las que luego será necesario hacer referencia en saltos, bifurcaciones, subrutinas, tablas, etc., .No obstante, todas las líneas pueden ser etiquetadas. El primer caracter de la etiqueta tiene obligadamente que ser una letra. Los caracteres posteriores, pueden ser tanto letras como números. La etiqueta no debe exceder los 13 caracteres. De ser de uno sólo, no puede ser la letra A, dado que esta la "A" puede indicar el modo de direccionamiento sobre el acumulador. Tampoco puede ser ninguno de los 56 mnemónicos del set de instrucciones del 6502.

CODIGOS DE OPERACION

Despues de la opcional etiqueta, sigue la parte más importante de la sentencia, llamada código de operación. Se puede tratar precisamente de un código de operación, un directivo o una macro definición. Antes de continuar, aclaremos estos términos: Código de operación es cualquiera de los mnemónicos del set 6502 tales como BRK, CLC, STA, etc. .Directivo o pseudoinstrucción, es una palabra reservada del ensamblador, para considerar los datos que le siguen de cierta manera, o para realizar una tarea compleja. Los directivos son descriptos al detalle más adelante. Una Macro es un nombre asignado a un grupo de instrucciones y su efecto es evitar el retipeo de las mismas instrucciones; es una comodidad para el programador. También, es detallada más adelante.
Efectuada la aclaración, digamos que de no existir etiqueta en la sentencia, el campo de instrucciones es la primera cosa que aparece en la línea. Si se trata de un código de operación o un directivo, el campo es muy breve, ya que los mnemónicos no tienen más de tres caracteres, y los directivos no más de cuatro. Las macro pueden tener hasta 13 caracteres.

OPERANDO / MODO DE DIRECCIONAMIENTO

El operando de una instrucción en código de máquina siempre declara el modo de direccionamiento de la misma. El micro 6502 (y sus sucesores) tiene varios modos de direccionamiento, pero no todas las instrucciones aceptan los mismos modos.
El SYSASM lo sabe y nos avisa con un mensaje de error toda vez que usemos una combinación no valida.
La sintaxis de los modos de direccionamiento es dada a continuación. Previamente definiremos el término "expresión". Una expresión es un conjunto de números, etiquetas y cálculos numéricos y lógicos que pueden ser evaluados para conseguir un resultado o valor. Visto de otro modo, una expresión es una fórmula que al evaluarse arroja como resultado un número. Bien, aclarado el término, lo abreviaremos EXPR.

Sintaxis	MODO DE DIRECCIONAMIENTO	Ejemplo
#EXPR	Inmediato.	LDA #10
EXPR	Absoluto, Relativo.	JSR OPEN
A	Acumulador.	ASL A
nada	Acumulador.	ASL
EXPR,X	Indexado por X, absoluto o página cero.	LDA TABLA,X
EXPR,Y	Indexado por Y, absoluto o página cero.	STA TABLA,Y
(EXPR,X)	Indirecto preindexado por X.	LDA (PNT,X)
(EXPR),Y	Indirecto postindexado por Y.	LDA (PNT),Y
(EXPR)	Indirecto.	JMP (VECTOR)

Como sabrá, si a leído algo sobre el 6502 (y sus sucesores), existe un modo de direccionamiento por página cero. El SYSASM no lo diferencia específicamente del absoluto, pero en el ensamble, si la instrucción puede expresarse en ese modo en vez del absoluto, la pone en modo página cero.

CAMPO COMENTARIO

Un comentario puede seguir a una sentencia en cualquier punto del programa. Para incorporarlo, lo separamos del resto de la línea con un punto y coma (;). No es necesario antecederle un espacio. Además, una línea puede llevar solamente este campo prescindiendo de los otros. Esta línea debe comenzar con un punto y coma y puede, si lo desea el programador, ir seguida de texto.

EXPRESIONES ASSEMBLER

Una buena parte de la potencia del SYSASM reside en el evaluador de expresiones. Esta es la sección de programa que calcula los valores de las expresiones, dadas en los operandos de las instrucciones. El SYSASM, para ayudarnos a abstraernos del programa lo máximo posible, permite usar complejas expresiones que involucran un amplio set de operadores y formatos de operando.
El SYSASM permite usar una expresión en todos los lugares que se requiera un valor, tal como en BASIC. No obstante, hay que recordar que el SYSASM trabaja con valores de 16 bits, más bien que en coma flotante como el BASIC. En adición, notará que el orden de evaluación de los términos en una expresión assembler, es estrictamente de izquierda a derecha, a menos que se usen paréntesis y estos no dependan del operando involucrado.
La expresión consta de términos y operadores. "Términos" son esencialmente valores, tal como números. "Operadores" son operaciones como suma, resta, etc.

FORMATO DE LOS TERMINOS

Un término en una expresión SYSASM puede tomar uno de los siguientes formatos:

Decimal: Se puede usar un número, tal como en BASIC. El número, debe estar en el rango 0 a 65535. No requiere prefijo. Ejemplo: LDA 10

Hexadecimal: Muchos programas ensambladores tienden a trabajar con números hexadecimales, puesto que éste es el que más refleja la naturaleza binaria de la máquina. Se puede usar números hexadecimales precedidos del prefijo $. No es obligado entrar los cuatro dígitos del número si este no lo requiere.
Ejemplo:JSR $FFD2

Octal: El octal no es una base numérica comunmente usada, pero se incorpora para guardar la compatibilidad con otros ensambladores que la incluyan. Se puede usar números octales precedidos del sufijo @. Tal como en hexadecimal, no es obligado poner ceros a la izquierda. Ejemplo: EOR @377

Binario: Las computadoras trabajan en binario, y a veces, es deseable entrar los números en este formato. Este podría ser el caso al usar operaciones lógicas como AND o ORA. Ideal, para el testeo de bits en componentes de entrada salida. Es posible utilizar números binarios precediéndolos del signo %. Los ceros a la izquierda son siempre opcionales. Ejemplo: AND %10000000

Contador de programa: Si lo desea, puede usar el valor del contador de programa como el valor del término. Esto suele ser muy útil al trabajar con una expresión relativa a la posición de una sentencia del programa; caso concreto: las bifurcaciones relativas. Se usa el asterisco (*) como símbolo. El asterisco toma el valor del contador de programa antes de que la actual instrucción se ensamble.
Ejemplo: BEQ *+3 ;saltear un byte.

Caracter: También es posible trabajar con caracteres ASCII. El caracter puede ir entre comillas dobles o simples. Ejemplo: LDA #"A" o LDA #'A'

Etiqueta: Tal como se vió en ejemplos previos, es posible usar etiquetas como términos. Dado el caso que tengamos un valor de frecuente uso en el programa, lo mejor es asignarle una etiqueta y usarla en lugar del valor. Esto será muy apreciado en el momento que, por alguna causa, tengamos que variar dicho valor. Bastará con cambiárselo a la etiqueta y el programa quedará igual. Esto nos recuerda claramente a las variables del BASIC.

Expresiones: Es posible utilizar una expresión completa como un término si se la encierra entre paréntesis. Esto nos permite cambiar el orden de la evaluación al calcularse la expresión. Por ejemplo: (#100+4) es un término válido. Damos otros ejemplos más adelante.

Negación: Podemos usar la negación de cualquier término, antecediendo el signo menos al mismo. Muy común al tener que usar números negativos decimales, no obstante podemos negar etiquetas y toda una expresión emparentizada, si fuese necesario.

Byte bajo/ Byte alto: Ud. puede tomar la parte baja "<" o la parte alta ">" de una expresión de 16 bits anteponiendo al término el signo menor o mayor respectivamente para cada caso. Puede aplicarse también a expresiones emparentizadas. Ejemplo: LDA <(LOOP+1+7)

Direcionamiento Absoluto Forzado: Como se mencionó previamente el SYSASM pone el modo de direccionamiento de una instrucción en modo página cero, si es posible. Normalmente, esto no causa problema en el programa, mientras no se manipulen los bytes de dirección de la instrucción. En caso de tener que sobre-escribir en ellas, será necesario forzar el direccionamiento absoluto para contar con low y high byte, y no tan sólo con low byte. Esto se consigue con el prefijo "!" antepuesto al término.

```
Ejemplo: TABLE=$0100
         LDA TABLA-1,X
```

La última sentencia será codificada en dos bytes. Si queremos que sea codificada en tres, la sentencia deberá tomar el siguiente aspecto:

```
LDA !TABLA-1,X
```

Operadores: Para hacer expresiones podemos combinar términos con operadores. Los operadores toman el operando a la izquierda y el de la derecha, tal como los operadores BASIC. En el SYSASM, los operadores son:

+	suma
-	resta
*	multiplicación
/	división
.	OR lógico
&	AND lógico
^	EOR lógico (XOR del basic)

Todos estos operadores toman valores de hasta 16 bits, pero con algunos de ellos sería absurdo usar números grandes. No tendría sentido multiplicar por un número mayor a 32767. Las multiplicaciones cuyo producto es mayor al contenido en 2 octetos (65535), generan un error de cantidad ilegal (Illegal Quantity Error). Los demás operadores no dan este error, puesto que la saturación (overflow) es a menudo enteramente legítima en el trabajo con complemento A dos aritmético.

Los siguientes ejemplos, demuestran que podemos combinar términos con operadores como nos plazca. No hay que olvidar que el evaluador de expresiones del SYSASM va de izquierda a derecha tal como se van encontrando los términos. Por ejemplo: el resultado de 5+4*3 en SYSASM es 27, puesto que la expresión, es evaluada así: (5+4)*3. En BASIC, el resultado sería 17, ya que su evaluador tiene una escala de jerarquía. Para lograr el efecto del evaluador basic use 5+(4*3), dado que el paréntesis se despeja primero, alterando el orden de la evaluación.

He aquí algunos ejemplos de expresiones para mostrar las posibilidades del evaluador de expresiones que incluye el SYSASM. El listado que sigue, como programa no hace nada y sirve únicamente como modelo de expresiones.

```
START   LDA TABLA-1+(TABLEN*2)
        STA $8000
        LDA #TABLA/8
        LDA #>START-1
        LDA #SIZE+>TABLA
        LDA #%1011+@3+$F
        LDA #-12
        LDA #(TABLA+2)
        LDA #<(ENTRADA+TABLA+128)
        LDA #TABLA+(ENTRADA*ENTSIZ)+OFF
        STA SCREEN+(LINEA*COLUMNA)+ACR
CODELEN=*
        LDA #>CODELEN
        LDY #<CODELEN
        JSR DOIT
MSG     .BYT "ESTE ES UN TEXTO"
MSGLEN=*-MSG
```

Ud. ahora tendrá alguna idea de cómo formar una expresión SYSASM. Puede usar toda la potencia del evaluador en el operando de cualquier expresión. Además, Ud., hallará muchos lugares donde poder usar expresiones. Asegúrese de aprovechar todas las ventajas de esta implementación. Algunos errores pueden surgir con el manejo de expresiones; Lea su significado en el apéndice de mensajes de error si incurre en uno de ellos.

DIRECTIVOS ASSEMBLER

Directivo	Función
=	Asignación a etiqueta
≪	Redefinir etiqueta
*=	Fijar contador de programa
*S=	Fijar contador de programa con valor relativo
.END	Fin del ensamble
.BYT	Reservar memoria para bytes
.ASC	Reservar memoria para bytes para caracteres
.POK	Reservar memoria para códigos de pantalla (PETASCII)
.WOR	Reservar memoria para palabras (low, high)
.DBY	Reservar memoria para doble-bytes (high, low)
.FIL	Llamar filas prg
.LIB	Llamar filas seq (librería)
.BIN	Incluir prg binario
.SST	Salvar tabla de símbolos
.IFE	Ensambla si el operando es 0
.IFN	Ensambla si el operando no es 0
.IFP	Ensambla si el operando es menor a 32768
.IFM	Ensambla si el operando es mayor a 32767
.ADD	Adicionar al código objeto
.EOR	Or exclusivo al código objeto
.SCS	Fijar suma de chequeo
.CCS	Dar salida a la actual suma de chequeo
.LPP	Fijar líneas por página
.PAG	Definir título de la página, saltar a la próxima
.SKI	Saltear líneas
.LIS	Listar
.NOL	No listar
.SYM	Listar tabla de símbolos ordenada alfabéticamente
.SYN	Listar tabla de símbolos ordenada numéricamente
.SYO	Listar tabla de símbolos en orden de aparición
.NOS	No listar tabla de símbolos
.INV	Entrar valores (input)
.INB	Entrar si/no (entrada booleana: yes/no)
.PRT	Emitir mensaje
.MAC	Definir macro
.MND	Fin de macro definición
.MEX	Listar la macro toda vez que se la llame
.MDX	No listar la macro al llamarla

Directivo: = (asignación a etiqueta)
Sintaxis: <etiqueta> = <expresión>
Descripción: El directivo "=" se usa para asignar un valor a una etiqueta o variable. El campo operando debe contener el valor numérico de la variable o una expresión evaluable por el ensamblador. El directivo "=" es ejecutado únicamente en el primer paso.

Directivo: « (redefinir etiqueta)
Sintaxis: <etiqueta> « <expresión>
Descripción: El directivo « se usa para asignar un valor a una etiqueta o variable que ya se ha definido. Si se pretende una redefinición con el directivo "=", se genera un error de redefinición. Si la etiqueta es redefinida con el directivo «, entonces no ocurre dicho error.

Directivo: *= (fijar contador de programa)
Sintaxis: *= <expresión>
Descripción: Define la dirección del ensamble. Cuando la dirección que señala es mayor a la última ensamblada, el espacio intermedio es llenado con ceros. No podemos poner una dirección anterior.

Directivo: *S= (fijar contador de programa relocalizado)
Sintaxis: *S= <expresión>
Descripción: Este directivo es similar al *=, excepto, que en las relocalizaciones los bytes salteados no son rellenados. El PC (contador de programa), aquí sí puede ir hacia atrás, en relación a su último valor asignado. El motivo de esto, podría ser un programa que requiere un cargador para correr en memoria baja. Por ejemplo: Al inicio del cargador se usa el directivo *=$0801 y después del mismo, se usa el directivo *S=$A000 direccionado como se ve en memoria baja (un concepto de C=64), y allí se desarrolla el programa. Después que el código objeto se carga, si el cargador terminase en $0850, entonces el código que corre en $A000, se hallaría situado en $0851 y requiere que el cargador lo tranfiere a su verdadera posición en $A000. Esto requiere sólo una fila y sólo un ensamble para operar. Podría hacerse con dos filas, pero esto es mucho más sencillo. Este modo de trabajo, es usado para escribir programas que corren dentro del drive; al final del código que corre en la computadora, Ud. podría tener un *S=$0300 y seguirlo con el código que corre dentro de la disquetera. Entonces en la ejecución el código que corre en la computadora podría cargar al código que corre en la disquetera.

Directivo: .END (fin de ensamble)
Sintaxis: [etiqueta] .END
Descripción: El directivo .END se usa para marcar el fin del ensamble. Cualquier sentencia escrita después del mismo será ignorada por el ensamblador. El directivo .END puede llevar una etiqueta, pero no usa operando (algún parámetro o argumento a la derecha). Su uso puede ser opcional, aunque se lo recomienda, a fin de un mejor listado y documentación.

Directivo: .BYT (reserva memoria con bytes)
Sintaxis: [etiqueta] .BYT <expresión> [,<expresión> ...]
Descripción: El directivo .BYT es usado para reservar un byte de memoria cargado con un valor. El directivo puede contener múltiples operandos los cuales almacenarán valores en bytes consecutivos. Cadenas de caracteres ASCII pueden ir encerradas en doble o simple comilla.

Ejemplos:

```
        INDICE .BYT 2
               .BYT 1,$F,%101,@7
        LETRAS .BYT 'ABCDEF',"FRED"
```

Note que los números pueden ser representados en el formato que más convenga. En general cualquier expresión válida cuya evaluación pueda ser contenida en 8 bits puede incluirse como operando del directivo. Si se desea incluir comillas en una cadena ASCII puede hacerse encerrándola con apóstrofos. Ejemplo: .BYT '"NOMBRE EXTERNO"'. La cadena se memorizará así: "NOMBRE EXTERNO". Si se quisiera incluir un apóstrofo en la cadena también puede lográrselo poniendo dos apóstrofos consecutivos. Verbigracia: .BYT 'JIM''S CYCLE' se usa para almacenar JIM'S CYCLE.

Directivo: .ASC (reservar memoria con caracteres)
Sintaxis: [etiqueta] .ASC <expresión> [,<expresión> ...]
Descripción: Es exactamente igual al directivo .BYT y sólo se incluye para fines de conversiones compatibles con otros ensambladores.

Directivo: .POK (reservar memoria para códigos de pantalla PETASCII)
Sintaxis: [etiqueta] .POK <expresión> [,<expresión ...>]
Descripción: Como el directivo .BYT, excepto que las cadenas de caracteres ASCII son almacenadas en PETASCII. Por ejemplo, una "A" en ASCII tiene el código $41, mientras que en PETASCII el código $01. Es muy conveniente para escritura directa sobre pantalla. Su nombre deriva de la sentencia POKE del BASIC, la cual suele utilizarse para los mismos fines.

Directivo: .WOR (reservar memoria para 2 bytes en el orden low, high)
Sintaxis: [etiqueta] .WOR <expresión> [,<expresión> ...]
Descripción: El directivo .WOR, es utilizado para reservar dos bytes de datos al mismo tiempo en formato low-high. Cualquier expresión numérica cuya evaluación pueda contenerse en 16 bits es válida como operando. Cabe aclarar que el valor a almacenar debe escribirse en su formato real (high-low), ya que la inversión será realizada por el directivo.

```
Ej: TABLA .WOR $FFD2    ;será almacenado $D2 $FF
          .WOR 1        ;se almacenará $01 $00
```

Esta forma de almacenamiento es la normal para cargar punteros, ya que estos señalan a una dirección en ese formato.

Directivo: .DBY (reservar memoria para 2 bytes en el orden high, low)
Sintaxis: [etiqueta] .DBY <expresión> [,<expresión> ...]
Descripción: Igual al directivo .WOR, sólo que los datos se almacenan en el orden high-low.

```
Ej: TABLA .DBY $FFD2    ;será almacenado $FF $D2
          .DBY 1        ;se almacenará $00 $01
```

Directivo: .FIL (encadenar filas)
Sintaxis: .FIL <fila fuente> [,D<x>[,S[,A]]] (x=número periférico del drive)
Descripción: .FIL pone fin al ensamble del archivo que contiene este directivo y comienza a leer código fuente del la fila del operando. No hay restricciones sobre la cantidad de filas que puedan enlazarse (linkearse) por el directivo. El parámetro "D" al final del directivo, selecciona el número periférico de la disquetera (por defecto se toma el dado por el comando DISK); Ej: D8, D9.
El argumento "S" al final del directivo sólo es válido cuando la fila corriente, es la de control o principal. Este parámetro hace que el ensamblador envíe un mensaje al usuario para cambiar de disco. Gracias a esto, el código objeto puede quedar en 2 o más filas. Las filas pueden ser concatenadas todas juntas o cargadas y corridas. Si tienen que ser cargadas, entonces, la dirección de carga debe asentarse en las otras filas y es esto precisamente lo que hace el parámetro "A" al final del directivo.
Los nombres de filas concomitantes son grabados así: FILA.1, FILA.2, etc., siendo FILA el nombre del archivo objeto. Si no se ensambla sobre fila objeto entonces no se da salida a los mismos.

Directivo: .LIB (filas librería)
Sintaxis: .LIB <fila fuente> [,D<x>] (x= periférico del drive)
Descripción: El directivo .LIB permite al usuario construir filas con rutinas de uso frecuente. Cuando un directivo .LIB es encontrado por el ensamblador, éste -temporalmente- detiene la lectura de la fila corriente y continúa leyendo de la fila en el operando del directivo. El procesamiento de la fila principal es retomado cuando se encuentra un .END o el fin del archivo del operando. Una fila de librería llamada por el directivo .LIB no puede contener este directivo, pero si puede llevar un .FIL (sólo 2 filas fuentes pueden estar abiertas simultáneamente). Ejemplo:

```
                    :
                    CONTROL                 SECUNDARIO
                    .LIB fila1 ----------> FILA1
                    <---------------------- .END
                    .LIB fila2 ----------> FILA2
                    .FIL fila3
                    FILA3
                    <---------------------- .END
                    .FIL fila4
                    FILA4
                    .END
```

Directivo: .BIN
Sintaxis: [etiqueta] .BIN programa [,D<x>[,A]] (x= periférico del drive)
Descripción: El directivo .BIN es como un gran sentencia .BYT. La fila es abierta y todos los valores binarios son leídos y enviados literalmente a la fila objeto. El procesamiento de la fila principal continúa una vez hallado el fin del programa binario. Este directivo es ideal para incluir: imágenes para pantalla, juegos de caracteres, sprites o todo tipo de tablas o datos almacenados en modo programa. El parámetro "A" al final del directivo hace que el ensamblador excluya los 2 primeros bytes del programa del operando. Esos 2 bytes son la dirección de carga del prg, y pueden no ser necesarios.

Directivo: .SST
Sintaxis: .SST <nombre> [,D<x>] (x=periférico del drive)
Descripción: El directivo .SST graba en disco durante el paso 1 del ensamble, la tabla de símbolos (todas las etiquetas y sus direcciones) en la fila del operando. La salvaguarda de las etiquetas se hace bajo el formato siguiente:
ETIQUETA=$VALOR. Cada línea termina con un retorno de carro y la última concluye con el directivo .END. EL directivo .SST graba todas las etiquetas que preceden su ocurrencia, de manera que para grabar todas las etiquetas debe estar tras la última línea donde haya una.

Directivos de ensamble condicional: .IFE, .IFN, .IFP, .IFM
Sintaxis: .IFx <expresión> [<]

```
.IFx
Código fuente opcional
>
```

Descripción: Los directivos de ensamblado condicional evalúan la expresión y si el valor corresponde a la condición del directivo (=0, <>0, <32768, >32767) el ensamblado continúa en la línea siguiente. Si la condición no es verdadera, el ensamblado continúa tras la línea siguiente al comando >. Si se usa etiquetas en la expresión, deberán haber sido definidas en el paso 1 del ensamble, antes de que el código fuente opcional sea procesado.

Directivo: .ADD
Sintaxis: .ADD <expresión>
Descripción: El directivo .ADD añade el valor de la expresión a todos los bytes objeto, hasta que otro .ADD sea encontrado. Por defecto es .ADD 0.

Directivo: .EOR
Sintaxis: .EOR <expresión>
Descripción: El directivo .EOR realiza un or exclusivo entre el valor de la expresión y cada byte objeto, hasta el próximo .EOR.

Directivo: .SCS
Sintaxis: .SCS <expresión>
Descripción: El directivo .SCS es usado para reponer el actual valor de la suma de chequeo (checksum). La suma de chequeo empieza en 0 y cada byte objeto pasa por un eor (or exclusivo) con el checksum previo. El propósito principal del directivo es poder ver si el programa ha sido modificado. Es un artificio de protección o un propio test de procedimiento. El actual valor de la suma de chequeo será enviado con el directivo .CCS como un byte. Usando un loop el checksum puede ser comparado con el verdadero checksum.

Directivo: .CCS
Sintaxis: .CCS
Descripción: El directivo .CCS envía al código objeto 1 byte conteniendo la actual suma de chequeo. (véase directivo .SCS).

Directivo: .LPP
Sintaxis: .LPP <número>
Descripción: Para impresión del ensamble por impresora, las líneas son automáticamente paginadas por el SYSASM. El formato de la página es: una línea en blanco seguida por el contador de página y el título en otra línea, la cual se continúa por otra línea en blanco. Tras todo esto viene el listado de las líneas fuente y tres líneas en blanco al final de la página. Las líneas por página -por defecto- son 66. Dado que se utilizan tres líneas de encabezamiento y tres para el margen inferior la cantidad de líneas fuente por página es: el número dado en el directivo, menos seis. Usando el directivo .LPP 66 (por defecto) tendríamos 60 líneas fuente por página.

Directivo: .PAG
Sintaxis: .PAG [título]
Descripción: Al tope de cada página (tras la línea inicial en blanco) sale el título de la página y el contador de las mismas. El título por omisión, es el nombre de la fila fuente de control. El directivo .PAG sin parámetro eyecta el papel hasta la próxima página y continúa la impresión. El directivo .PAG con título eyecta el papel al tope de la próxima página y redefine el encabezamiento del nuevo folio. La longitud máxima del título es de veinte caracteres.

Directivo: .SKI
Sintaxis: .SKI [número]
Descripción: El directivo .SKI indica a la impresora cuántas líneas debe dejar en blanco. Sin parámetro se asume que es 1. El directivo .SKI, es ideal para separar subrutinas en el listado, sin destinar una página para cada una.

Directivo: .LIS
Sintaxis: .LIS
Descripción: El directivo .LIS (activo por defecto) permite imprimir las líneas fuentes que le siguen hasta encontrar un directivo .NOL

Directivo: .NOL
Sintaxis: .NOL
Descripción: El directivo .NOL suspende la impresión del listado hasta encontrar un directivo .LIS. Ambos directivos son prácticos ya que combinados, permiten listar secciones de la fila fuente.

Directivo: .SYM
Sintaxis: .SYM
Descripción: Activo por defecto el directivo .SYM lista la tabla de símbolos alfabéticamente, al final del paso 2 del ensamble.

Directivo: .SYN
Sintaxis: .SYN
Descripción: El directivo .SYN lista la tabla de símbolos numéricamente (veces que se ha invocado una etiqueta), al final del paso 2 del ensamble.

Directivo: .NOS
Sintaxis: .NOS
Descripción: El directivo .NOS prohibe el listado de la tabla de símbolos.

Directivo: .INV
Sintaxis: .INV <etiqueta>
Descripción: El directivo .INV recibe un valor por parte del usuario durante el paso 1 únicamente del ensamble. El valor puede estar en cualquier base numérica o en ASCII. Si el valor es incorrecto es repedido hasta que sea correcto.
Ver ejemplo dado en el directivo .PRT más adelante.

Directivo: .INB
Sintaxis: .INB <etiqueta>
Descripción: El directivo .INB recibe una Y o una N de parte del usuario y deja un 1 por la "Y" o un 0 por la "N" en la etiqueta, sólo durante el paso 1. Al dar entrada sin ningún valor, se asume una "Y".
Ver ejemplo dado para el directivo .PRT siguiente.

Directivo: .PRT
Sintaxis: .PRT <"texto ...">
Descripción: El directivo .PRT envía un mensaje durante el paso 1 al usuario. Un retorno de carro sigue al mensaje. Ejemplo:

```
.PRT "Indique tipo de computadora:"
.PRT "1 = Commodore 16"
.PRT "2 = Commodore 64"
.PRT "3 = Commodore 128"
.PRT "4 = Commodore Amiga"
.INV TIPMAQ     ;entrar valor en etiqueta TIPMAQ
.PRT "Formato 80 columnas (y/n)?"
.INB PANTALLA   ;entrar tipo de pantalla en esa etiqueta
```

El ejemplo previo permitiría a una fila fuente ser capaz de ensamblar un programa con versiones diferentes para cada máquina sin tener que usar un programa de control distinto para cada una. La fila fuente usando los directivos condicionales puede generar distintos códigos objeto independientes cuando sea requerido.

Directivo: .MAC
Sintaxis: .MAC <nombre de la macro>
Descripción: El directivo .MAC es usado para permitir la definición de macros. Una macro, debe ser definida antes de que pueda ser llamada o incluida por otra. Las líneas de texto después del directivo .MAC son almacenadas en el área de macrodefiniciones hasta que el directivo .MND sea encontrado.

Directivo: .MND
Sintaxis: [etiqueta] .MND
Descripción: El directivo .MND es usado para marcar el final de una macrodefinición. La línea puede contener una etiqueta la cual será parte de la macro actual (vease el directivo .MAC).

Directivo: .MEX
Sintaxis: .MEX
Descripción: El directivo .MEX le dice al ensamblador que imprima la macro línea a línea, hasta que un directivo .MDX aparezca.

Directivo: .MDX
Sintaxis: .MDX
Descripción: El directivo .MDX le dice al ensamblador que cuando encuentre una macro, sólo liste el nombre y los parámetros de la misma, hasta que un directivo .MEX aparezca.

CAPACIDAD DE UNA MACRO

Una macro es esencialmente una síntesis, por la cual, una línea de código fuente puede representar una función consistente de muchas secuencias de instrucciones. Por ejemplo el set de instrucciones del 6502 (y sus sucesores) no tiene una instrucción que incremente una dirección de memoria de doble byte. Podríamos escribir una macro que ejecutara esta operación y que podríamos representar con INCD. Esta macro aparecería en el listado assembler en el campo de los mnemónicos, de modo similar al que sigue:

```
STA TEMP
INCD TEMP1 ;macro incremento de doble byte
```

Pero antes que la macro pueda utilizarse debe ser definida. El nombre de la misma no podrá ser ninguno de los mnemónicos del set del 6502, ni tampoco el de una etiqueta existente. La sintaxis para el nombre de la macro es la misma que para las etiquetas. Las macrodefiniciones toman el formato que sigue:

```
.MAC <nombre de la macro>
texto de la macro
[etiqueta] .MND
```

El directivo .MAC define una macro con el nombre dado. El usuario declara explícitamente parámetros. La macro puede contener texto arbitrario pero una macro no puede ser definida dentro de otra macro.
El ensamblador no genera el código objeto de la macro hasta que esta es llamada. para llamar a una macro, el usuario simplemente da el nombre de la misma seguida de sus parámetros según este formato: NOMBREMACRO param1:param2:...

El nombre de la macro debe separarse con por lo menos, un espacio del primer parámetro, tal como se indica en el formato.
En el texto fuente de la macrodefinición, un parámetro es designado por el símbolo "?" seguido por un número. Así "?3" hace referencia al tercer parámetro. La cantidad máxima de parámetros está limitada por la capacidad de la línea fuente: 80 caracteres para C=64 y 160 para C=128. Durante el ensamble si se encuentra un llamado a una macro, el texto fuente de esa macro es insertado en el ensamble en ese punto y los parámetros, sustituidos por los nombres temporales allí dados.
Si el usuario omite un parámetro al llamar la macro, el ensamblador lo sustituye por una cadena nula. Se asume que las etiquetas definidas dentro de una macro, son para uso propio de la misma. Si son definidas etiquetas globales dentro de la macro, se antecede el prefijo "+" al nombre de la misma. Entiéndase por "etiqueta global" a una etiqueta que luego, va ha necesitarse en otra parte del programa FUERA de la macro.

PASANDO PARAMETROS A UNA MACRO

Los parámetros van separados por dos puntos (:). Si dentro de un parámetro, hubiesen dos puntos (:), pero estuvieran entre simple o doble comilla, no se los considera como marca de fin de parámetro. Los parámetros que se omiten se declaran por dos puntos concomitantes sin texto intermedio:

```
        FUNCION aa:::dd:      ;se omiten los parámetros 2 y 3.
        FUNCIONX :::cc::ee:   ;se omiten los parámetros 1, 2 y 4.
```

Un parámetro no debe llevar espacios espacios intermedios. Se pueden pasar parámetros que sean modos de direccionamiento, expresiones, etiquetas, texto o cualquier otra cosa. Ejemplo de parámetros:

```
    ADD (prt),y:#$02:          ;parám 1 modo de direcc., el 2 un valor.
    OPEN15:8:15:"I0":          ;ejecuta el comando OPEN 15,8,15,"I0"
    PRINT 147,13,13,"menu",13: ;imprime un texto
```

Si lo desea puede poner un comentario tras los parámetros antecedido por el signo ";". De otro modo, se considerará el comentario como parte del último parámetro. He aquí algunos ejemplos de macros:

```
            .MAC PRINT
            LDY #0
    IMPRIMA LDA TEXTO,Y
            BEQ FINTEXT
            JSR CHROUT
            INY
            BNE IMPRIMA
    TEXTO   .BYT ?1,0       ;parámetro ha pasar
    FINTEXT .MND            ;texto impreso
```

He aquí cómo llamar a la macro PRINT:

```
    PRINT 147,13,?titulo?:  ;se emitirá el texto dado
```

Demos otro ejemplo de macro:

```
            .MAC DOPEN
            LDA #?1
            LDX #?2
            LDY #?3
            JSR SETLFS              ;Etiqueta global ya definida (no propia.)
            LDA #ENDNAME-NAME       ;calcular longitud del nombre
            LDY #>NAME              ;tomar parte alta de dirección del nombre
            LDX #<NAME              ;tomar parte baja de dirección del nombre
            JSR SETNAM
            JSR OPEN
            JMP ENDNAME
    NAME    .BYT ?4                 ;Nombre del archivo a abrir
    ENDNAME .MND
```

He aquí como llamar a la macro DOPEN:

```
    DOPEN 15:8:15:"I0":     ;Inicializar disketera.
```

MENSAJES DE ERROR DEL ASSEMBLER

ERRORES NO FATALES (el ensamble continúa)

UNDEF LABEL	(no se a definido la etiqueta)
REDEF ERROR	(se pretende definir una etiqueta ya declarada)
ILL OPCODE	(código de operación ilegal, no existe esa instrucción)
ILL DIRECTIVE	(no existe ese directivo)
ILL ADDR MODE	(no se aplica ese modo de direccionamiento)
ADDR MODE MISTMATCH	(no existe esa combinación. Ej: jsr ($ff),y)
ILL QUANTITY	(valor muy grande. Ej: .byt 256)
LINE TOO LONG	(la línea excede el límite del buffer de entrada)
LABEL TOO LONG	(nombre de etiqueta con más de 13 caracteres)
ILL START CHAR	(La línea comienza con un caracter equivocado)
OUT OF RANGE	(bifurcación mayor a 127 o menor a -128)
MISSING COMMA	(faltó una coma)
UNBALANCED PAREN	(paréntesis desbalanceados, sobran o faltan de un lado)

Errores fatales (se detiene el ensamble)

NO END OF .IFX ERR	(no se halló fin del ensamblado condicional)
DISK ERR	(error de entrada/salida)
PC REVERSAL	(contador de programa en sentido retrógrado)
SYMBOL TABLE OVERFLOW	(área de almacenamiento para etiquetas saturada)
MACRO MEMORY OVERFLOW	(área de almacenamiento para macros saturada)
MACRO STACK OVERFLOW	(excesivos anidamientos en la macro)

COMENTARIOS SOBRE LOS ERRORES DURANTE EL ENSAMBLE

Errores no fatales (el ensamble continúa)

Undefined label

Este error se genera en el segundo paso si una etiqueta en una expresión no ha sido definida.

Redefinition error

Este error se genera si se pretende definir una etiqueta con el directivo "=" después que ha sido asentada en la tabla de símbolos. Si verdaderamente se necesita redefinir la etiqueta, se lo hace con el directivo «.

Illegal op-code

El assembler busca en la línea fuente, la primera ocurrencia de un caracter distinto a "space" y forma una cadena hasta la siguiente ocurrencia. Si la cadena conformada no es un código de operación o una macro, cree que se trata de una etiqueta y la declara en la tabla de símbolos. De todos modos, si encuentra un segundo campo, lo considera campo del código de operación, (dado que no puede haber más de una etiqueta por línea fuente). Si la nueva cadena formada no es un código de operación que exista, el mensaje de error es impreso.

Illegal Directive*

Todos los directivos del ensamblador comienzan con un punto. Si un punto es el primer caracter en un campo no vacío, el ensamblador interpreta la cadena de caracteres que le siguen como un directivo. Si dicha cadena no es un directivo assembler válido, este mensaje de error será impreso.

Illegal Address Mode

Si la cadena del código de operación es válida y el modo de direccionamiento también, pero ambos juntos no crean un instrucción legal, entonces ocurre este error. Ejemplo: jsr ($0090). El código jsr no tiene direccionamiento indirecto, por lo tanto es generado el mensaje correspondiente.

Illegal Quantity

El valor de una expresión es muy grande. Ejemplo: .BYT 257. En el ejemplo el valor excede la capacidad de un byte, la cual va de 0 a 255.

Line too long

La línea leída por el assembler tiene más caracteres que la longitud del buffer de entrada: (80 en modo 64 y 160 en modo 128). Debe descomponerla en más segmentos.

Label too long
Todas los nombres de etiqueta se limitan hasta 13 caracteres de longitud. Cuando analiza, el ensamblador busca un caracter no alfanumérico para encontrar el fin del nombre de la etiqueta. Si la cuenta pasa los 13 caracteres, este mensaje de error es emitido.

Illegal start character
El primer caracter en una línea debe ser alfabético (a-z) para indicar una etiqueta o un código de operación. O puede ser un punto "." para indicar un directivo. También puede ser un asterisco "*" para referirse al contador de programa: Por último se acepta el signo mayor ">", como referencia al fin de un ensamblado condicional. Cualquier otro caracter inicial causará este error.

Out of range
Todas las instrucciones de bifurcación son ensambladas en dos bytes objeto. Un byte para el código de operación y el otro para la dirección a la que se bifurca. Para permitir saltos relativos hacia adelante o hacia atráś indicados con un sólo byte, todos los valores entre 0 y 127 son hacia adelante y los comprendidos entre 128 y 255 son hacia atrás. Si la dirección a donde se pretende bifurcar está más allá de este limite, es generado este error.

Missing comma
Se esperaba una coma, pero otro caracter fue hallado.

Unbalanced paren
Una expresión tiene un número desigual de paréntesis.
Ejemplo: 12+(12*23))

ERRORES FATALES (se detiene el ensamble)

No end of .ifx
Si mientras el assembler está procesando un directivo condicional, el resultado de la expresión condicional es falso, el ensamblador saltea las líneas siguientes hasta encontrar un signo mayor ">" que indique el fin de ese ensamble condicional. Si el fin de archivo es encontrado antes de esta ocurrencia, se genera el error.

Disk error
Si leyendo un texto fuente o grabando un código objeto un error de entrada/salida se avisa por el sistema operativo, el nombre del actual fichero es visualizado en pantalla, junto con el mensaje del DOS.

PC reversal
Debido a que la salida del código objeto se asienta en una fila tipo prg, el contador de programa sólo puede ir en un sentido: hacia adelante... Si intenta que el PC corra en una dirección previa a la actual, después de iniciarse la salida del código objeto, se da este error.

Symbol table overflow
Los símbolos o etiquetas requeridos en el programa son ordenados internamente con sus valores. Si la memoria que se les reserva se agota, caemos en este error. La soluciones pueden ser: Partir el programa en más filas fuentes; remover etiquetas innecesarias; acortar la longitud de sus nombres; borrar los espacios innecesarios y eliminar comentarios.

Macro memory overflow
La memoria reservada para las macro es de 8Kb. Las macro se almacenan secuencialmente en su formato textual sobre su área. Si se usa toda esa memoria se genera este error. Si esto sucede elimine todos los espacios innecesarios y los comentarios lo máximo posible de las macro.

Macro stack overflow
La única limitación para el anidamiento de macros, es el tamaño de la pila de macros. Cada vez que una macro invoca a otra, los nombres de las macros y sus parámetros son salvados en la pila. Por eso, la cantidad y longitud de los parámetros son el real límite de anidamiento de macros. Si el espacio requerido para salvar los valores de las macros que se anidan no alcanza se da este error.

EDITOR DE SPRITES TSDS [SPED]

Con este editor de sprites se puede editar hasta 256 sprites a la vez. Los sprites son salvados en disco en formato programa. Cuando se carga un programa de sprites, no importa en que bloque del VIC fue salvado porque el editor lo reloca-liza en el bloque 1. Y cuando se salva un bloque de sprites, se ingresa el número de bloque y la dirección de carga se altera apropiadamente.

Teclas de edición

Todas las ediciones del cursor son tomadas desde el teclado o del joystick 2.

RVS = Invierte sprite.
CLR = Borra sprite.
HOME = Casa del cursor (esquina superior izquierda).
RETURN = Nueva línea, en extremo izquierdo.
. = Enciende un punto.
BARRA = Borra un punto (*).
← = Invierte un punto (*).
FUEGO = Encender, Borrar, Invertir un punto, según el modo seleccionado (*).

(*) En modo multicolor, incrementa el valor del color (0-3).

COMANDOS DEL MENU PRINCIPAL

+	Ir al próximo sprite.
-	Ir al sprite anterior.
g	Entrar el número de sprite al que se desea ir.
a	Modo animado.
A	(shift a), desactivar modo animado.
c	Menú de cambio de color.
f	rellena un bloque de sprites (con rotación).
i	Mezcla sprites.
j	Acción del fuego del joystick 2.
k	Desplazamiento automático del cursor.
l	Cargar.
m	Conmutador multicolor/monocolor.
o	Conjunción de sprites.
O	(shift o) desactivar conjunción de sprites.
p	Postura del sprites.
s	Grabar.
t	Transferencia de rangos.
v	Visualización.
z	Fijar grados de la coordenada central (eje central).

DESCRIPCION DE LOS COMANDOS

Dado que algunos comandos son auto-explícitos, sólo se comentan aquellos que son más complejos.

(a) ANIMACION.

El modo de animación se usa para mostrar más de un sprite en la misma posición, uno por vez. Por ejemplo: Una secuencia de sprites que muestra cómo vuela un pájaro, cuando vamos pasando de un sprite a otro a una velocidad constante. Los sprites en este caso, deben estar dispuestos en orden secuencial. Existen dos modos de movimiento: uni-direccional y bi-direccional. La velocidad, es el retardo entre la visualización de uno al sprite siguiente. Cada unidad de velocidad está expresada en "jiffies" (1/60 de segundo).
Ejemplo: si el rango de sprites fue desde el 0, al 5:
Unidireccional: 0,1,2,3,4,5,0,1,2,3,4,5,0,1,2,...
Bidireccional: 0,1,2,3,4,5,4,3,2,1,0,1,2,3,4,...

(c) MENU PARA CAMBIO DE COLOR.

Dentro de este menú mediante F1 se retorna al menú principal. Los colores que se pueden cambiar son: COLOR PROPIO DEL SPRITE, COLOR DE FONDO, COLOR DE BORDE, COLOR SPRITE MULTICOLOR 0 y COLOR SPRITE MULTICOLOR 1. Para cambiar un color, primero seleccione cuál, presionando la primera letra de la palabra y luego use las combinaciones (control 1-8) o (commodore 1-8) para fijar el nuevo color. Ej: para poner el borde de color blanco, pulse E y luego CONTROL 1.

(f) RELLENAR UN BLOQUE.

Este comando hace copias de un sprite, en un rango de sprites. Por ejemplo si se quiere hacer cinco pescados pero cada uno ligeramente distinto del otro, una vez dibujado el primero digamos en el sprite 0, se puede aprovechar el comando y entrar: SPRITE ORIGINAL= 0, DESDE= 1, HASTA= 4, ROTACION= 0. Entonces se podrá modificar los sprites 1-4 a gusto. Opcionalmente es posible rotar el bloque que se copia en cualquier número de grados, ingresando su valor al responder al parámetro ROTACION.

(i) MEZCLAR SPRITES.

Este comando permite poner dos sprites juntos y mezclar sus imágenes. Sólamente, hay que indicar cuál es el número de sprite que ha de fusionarse con el actualmente visualizado.

(j) ACCION DEL FUEGO DEL JOYSTICK 2.

El bptón de disparo puede: invertir, borrar o encender un punto en la actual posición del cursor. Para seleccionar la función tipee el primer caracter del nombre del modo. Ej: para seleccionar el modo encender (SET) tipee "s".

(k) DESPLAZAMIENTO AUTOMATICO DEL CURSOR.

Cuando se edita un sprite, el cursor puede: permanecer donde está o moverse a su derecha, tras la última acción sobre los puntos (inversión, fijación o borrado). Para cambiar de modo (fijo o movil) presione "k" desde el menú principal y pasará de uno a otro estado.

(l) CARGAR.

Este comando muestra un directorio de todos los programas del disco basado en el patrón de búsqueda entrado y le permite elejir cuál va a cargar. Si la longitud del nombre del programa excede los 10 caracteres, el resto no se visualiza. Para seleccionar el programa a cargar se utiliza las teclas del cursor arriba/abajo y en modo inverso van listándose las filas. Con RETURN, se carga la fila en inverso.

(o) CONJUNCION DE SPRITES.

Este comando se usa para mostrar otros sprites alrededor del sprite en edición y está principalmente destinado para la formación de figuras que abarcan más de un sprite. Puede juntarse hasta seis sprites a un tiempo. Para hacer una conjunción se ingresa el número de conjunción (1-6) y el número de sprite (0-255). Luego se posiciona el sprite sobre la pantalla con las teclas del cursor o el joystick, y se pulsa F1 cuando está listo.
Con el comando SHIFT-O, se elimina una conjunción. Simplemente se ingresa el número de conjunción a eliminar (1-6) y se da entrada.

(p) POSTURA DEL SPRITE.

El ingreso a este sub-menú nos permite mover la imagen del sprite en todo sentido. Notará que los comandos de scroll arriba, abajo, izquierda y derecha, tienen dos modos diferentes de ejecución. En modo mayúscula (U/D/L/R) rotan el sprite y en modo minúscula (u/d/l/r) lo desplazan, perdiendo las partes que escapan de la grilla. Aunque la grilla de un sprite es de 21 'y' por 24 'x' una grilla de 24 x 24 se preserva hasta que se sale de este sub-menú.

Comandos de postura del sprite

U/u	scroll hacia arriba.
D/d	scroll hacia abajo.
L/l	scroll a la izquierda.
R/r	scroll a la derecha.
x	espejo horizontal.
y	espejo vertical.
s	grados de rotación (1-359).
F1	vuelta al menú principal.

(s) GRABAR.

Para grabar un bloque de sprites, se ingresa el nombre con el que se lo salva en disco -máximo 9 caracteres-. El sistema, antepone automáticamente "0:" al nombre dado. Luego, debe indicarse cuál bloque se desea grabar (0-3), número inicial, y número de final de sprite inclusive. Con estos datos, se produce la grabación.

(t) TRANSFERENCIA DE RANGOS.

Para copiar un bloque de sprites en otro, simplemente indique: número inicial de sprite y número de sprite final de origen. Luego, el número de sprite inicial de destino. Hecho esto, de entrada y se ejecuta el comando.

(v) VISUALIZACION.

Este comando nos permite visualizar 64 sprites al mismo tiempo. Para elejir cual bloque de sprites se muestra, simplemente ingrese su número (0-3). De inmediato, se genera la imagen. Pulse cualquier tecla para volver al sub-menú. Pulse F1 para volver al menú principal.

DESENSAMBLADOR (UNASSEMBLER)

Un desensamblador, es un utilitario que convierte un programa objeto -escrito en lenguaje de máquina- a su código fuente. Pero, el desensamblador necesita alguna información adicional aparte del nombre del programa a desensamblar. Esto es, la locación de bloques de texto, tablas de bytes, tablas de palabras (words), áreas de código, etc.
Toda esta información debe ser ingresada por el programador. Puede adquirirla si se vale de un monitor (tal como el STARMON) y lista el código del programa a desensamblar. Gracias al monitor, puede conformar una hoja de datos donde anote el inicio y fin de cada tipo de dato.
Una vez conseguida la información, el desensamblador está en condiciones de sernos útil. Lo primero que nos pide es el nombre del programa a desensamblar. Luego, nos solicita la información que conseguimos con el monitor. Supangamos que nuestra hoja de datos llegó al siguiente resultado:

Tipo de información	Inicio	Fin
Código	C000	C080
Bytes	C081	C090
Texto	C091	C230
Código	C231	C5E2
Words (bytes low-high)	C5E3	C5FF

El desensamblador nos pide que entremos estos datos en el siguiente formato: Una letra nos dice la naturaleza de la información. "C" codigo, "B" bytes, "T" texto "W" words. Es lo primero que se indica y se separa con una coma, tras la cual se ingresa el inicio en hexadecimal, otra coma y la dirección final tras lo cual se da entrada. Esto se repite hasta que hemos acabado de entrar los datos. Para señalarlo, damos entrada a la letra "F".
El archivo fuente resulta siempre mayor a su par objeto, por lo que el desensamblador nos consulta sobre la cantidad de bloques en que particionaremos la salida del código fuente. El desensamblado podrá almacenarse así en varios archivos, de no alcanzar con uno, todos con no más del máximo de bloques indicado. El nombre de la primera (o única) fila fuente es solicitado y tras su ingreso, se inicia la tarea. El resultado quedará alojado en el diskette bajo el siguiente formato, suponiendo que "NOMBRE" fuese el nombre de la fila fuente:

NOMBRE	fila de control para reensamble posterior.
NOMBRE.LAB	fila que contiene todas las etiquetas y valores.
NOMBRE.00	primer fila fuente.
NOMBRE.nn	última fila fuente.

El desensamblador cambia todas las posiciones de memoria usadas (tal como $FFD2) por una etiqueta con el formato ZFFD2. De hecho, antepone una "Z" a los valores hexadecimales.

Ahora, con el SYSASM, puede editar el programa, adaptarlo a sus necesidades y si está correctamente desensamblado, re-ensamblarlo con sus modificaciones.

DEPURACION DEL DESENSAMBLADO

Si desea cambiar todas las etiquetas con formato Zxxxx por otras más significantes, siga este procedimiento:

1) Cargue el SYSEDT.
2) Con el comando GET, cargue una fila fuente. (inclusive la .LAB).
3) Ponga el disco TSDS y tipee el comando EXEC LABELS.
4) Ahora regrabe la fila fuente cargada, en su diskette original.
5) Repita los pasos 2 a 4 hasta acabar con todas las filas.

Lo que hace este programa es ejecutar un conjunto de comando CHANGE para cambiar etiquetas como ZFFD2 por CHROUT (su verdadero nombre).

INSTRUCCIONES SOBRE EL CONVERSOR

El programa CONVERTER se usa para convertir código fuente de otros ensambladores tales como PAL y MAE, para ser usado por el SYSASM del TSDS. Es muy simple de operar y hace un trabajo de conversión bastante bueno.

Tras cargar el conversor, hágalo correr y siga las instrucciones en pantalla.

EDITOR DE EFECTOS SONOROS (SOUND FX EDITOR)

Gracias a Shelly Kantrow y David Miller (autores de "3001 a Sound Odessey") contamos con esta original rutina de efectos sonoros.

Con este editor de efectos sonoros podemos editar un conjunto de hasta 256 efectos por vez. Los efectos son almacenados en 2 programas. La primera fila se llama "E.NOMBRE" y contiene trece bytes de datos por cada efecto. La segunda se denomina "N.NOMBRE" y guarda 16 bytes por efecto los cuales son el nombre del mismo. Ahora bien, cuando se carga un programa de efectos, el editor se ubica automáticamente sobre el efecto número 0.

El editor de efectos opera bajo el control de una rutina en lenguaje de máquina. El código fuente de esta rutina se encuentra en el diskette del TSDS con el nombre "SNDEFFS.S". Por su parte, el editor de sonidos SOUND FX EDITOR permite crear efectos que luego pueden ser añadidos al código fuente de la rutina y ser incoporados a nuestros propios programas.

Pueden ejecutarse hasta 3 efectos a la vez. Si las tres voces del chip de sonido están ejecutando un efecto, se ignoran los llamados a otros efectos hasta que se desocupe alguna voz.

Estructura de los efectos

Cada efecto es un set de 13 bytes de datos. La siguiente información es usada en la generación habitual de sonidos en el chip SID:

1) Frecuencia
2) Ancho de pulso
3) Ataque
4) Sosten
5) Decaimiento
6) Liberación
7) Forma de onda

En adición a esta información, el editor usa un bloque de información del efecto consistente en REPETICION (0-15), ENVOLVENTE e INTENSIDAD.

El envolvente ADSR es para cambiar la frecuencia del efecto. La intensidad, para fijar precisamente la intensidad del seguidor de envolvente. Mientras más se sube la intensidad, más cambia la frecuencia en relación con el seguidor de envolvente ADSR.

Teclas de edición

Todos los parámetros del efecto son accedidos por las teclas del cursor.

">" o "."	Incrementa la frecuencia o el pulso.
"<" o ","	Decrementa la frecuencia o el pulso.
"p"	Ejecuta el efecto.
"n"	Da un nombre al efecto.
"+"	Pasa al siguiente efecto.
"-"	Va al efecto anterior.
"g"	Ir a un efecto (0-255).
"l"	Cargar un conjunto de efectos.
"s"	Grabar un conjunto de efectos.

Descripción de los comandos

(p) EJECUTAR UN EFECTO.

Este comando ejecuta el actual efecto en edición. Si no se produce ningún sonido chequee atentamente los parámetros ingresados y asegúrese de fijar la frecuencia adecuada. En caso de usar la forma de onda de pulso ponga un valor en este parámetro. Si bien no se requiere una formación musical, usted al menos debería contar con nociones básicas sobre el SID y su programación al pretender crear efectos.

(n) NOMBRAR UN EFECTO.

Este comando permite que todos los efectos tengan un nombre de hasta 16 caracteres. De este modo, contamos con un sistema mnemotécnico que de un golpe de vista nos permite recordar la naturaleza del efecto creado.

(l) CARGAR UN CONJUNTO DE EFECTOS.

Este comando muestra un directorio selectivo de programas-archivo de efectos. Usa el mismo sistema de selección que el editor de sprites. Refiérase al mismo.

(s) GRABAR UN CONJUNTO DE EFECTOS.

Este comando trabaja exactamente como el grabador del editor de sprites. Tenga a bien dirijirse al mismo para su descripción.

COMO INCORPORAR EFECTOS A SUS PROGRAMAS

Para agregar efectos a sus programas proceda como sigue:

1) Carge el SYSASM del TSDS.
2) Tipee GET SNDEFFS.S
3) Cambie en la línea 1480 sobre el directivo .BIN el nombre del programa de efectos por el suyo.
5) Ponga su propio diskette y grabe el código fuente así modificado, y genere el código objeto.
6) Para testear el programa, use BLOAD y cargue el código objeto. Active el modo BASIC del editor con EDIT B y entre el comando NEW.
7) Ahora puede escribir este corto programa de test en BASIC, como sigue:

```
10 SYS 49152     :REM INICIALIZAR RUTINA DE EFECTOS
20 INPUT "NUMERO DE EFECTO";E
30 POKE 49152+4,E :REM FIJAR NUMERO DE EFECTO
40 SYS 49152+3    :REM EJECUTAR EFECTO
50 GOTO 20
```

Si desea incorporar efectos sonoros a su programa en assembler, puede usar hacer uso de esta rutina. Primero antes que todo debe inicializar el SID con el llamado a la subrutina INITSID, así: JSR INITSID. Luego, en el resto del programa debe indicar en el acumulador el número de efecto y llamar a la subrutina PLAYEFF, de este modo por ejemplo:

```
LDA #1              ;NUMERO DE EFECTO A EJECUTAR
JSR PLAYEFF         ;EJECUTARLO
```

Además, observe que la etiqueta ZPSND ni la direeción que ocupa sea utilizada en su programa para otros fines. En el caso de estar usando su propia rutina de interrupción, debe llamar directamente dentro de la misma a la rutina EFFIRQ y borrar el código que desvía el vector irq dentro de la rutina INITSID.

EDITOR DE PANTALLAS Y CARACTERES

Mapa de memoria del MATTE/CHRSET EDITOR [MACSED] -su nombre original-:

$0801 a $2FFF	Macsed
$3000 a $3FFF	Los 2 juegos de caracteres
$4000 a $49FF	Variables
$4A00 a $7800	Editor
$D100 a $FF00	Color

Dado el mapa de memoria, el SYEDT/SYSASM editor/ensamblador puede residir simultaneamente con el MACSED.

MACSED es un programa que permite editar juegos de caracteres y con ellos, generar pantallas. El propósito principal del Macsed es la creación de pantallas para distintos niveles en los juegos, aunque sus aplicaciones son infinitas.
El programa arranca en el editor de pantallas [Matte editor]. Pulsando F7 entramos al menú principal. Desde allí, pulsando F1 podemos pasar al editor de caracteres [Chrset editor].

EDITOR DE PANTALLA

La edición se realiza con las teclas del cursor. En el borde superior de la pantalla, hay una línea que sirve de referencia entre el juego de caracteres normal y el actualmente cargado. El editor de pantallas las almacena en disco en bajo 2 programas. El primero es llamado "B.NOMBRE" e indica en el primer byte, el ancho de la pantalla; en el segundo, el largo, lo cual nos indica el tamaño de la pantalla. Estos dos bytes sirven para saber cómo deben distribuirse el resto de los bytes sobre la misma. Un segundo programa llamdo "B/NOMBRE" comienza con los indicadores descritos y luego sigue la información de color.

Teclas de edición

cursor arriba	Sube una línea.
cursor abajo	Baja una línea.
cursor izquierda	Retrocede una columna.
cursor derecha	Adelanta una columna.
home	A la esquina superior izquierda actual.
home home	A la esquina superior izquierda inicial.
insert	Insertar un caracter.
delete	Borrar un caracter.
teclas de color	Elije color para escribir.
return	Va al borde izquierdo del Matte.
shift return	Va al borde izquierdo de la pantalla visualizada.
F1	Scrola línea de referencia a la izquierda
F3	Scrola línea de referencia a la derecha.
F5	Incrementa color del fondo.
F7	Al menú del editor.

Menú del editor

F1	Ir al editor de carateres.
l	Cargar una pantalla.
s	Grabar una pantalla.
r	Fijar un rango.
c	Copiar un rango.
t	Transferir un rango.
e	Borrar todo o borrar un rango.
b	Definir blanco.
w	Definir ventana.
insert	Insertar una línea.
delete	Borrar una línea.

Comandos del editor de pantalla

(l) Cargar una pantalla.

El cargador de pantalla nos consulta sobre un patrón de carga. Entonces, genera un directorio selectivo en relación al patrón dado. La selección de la pantalla a cargar se realiza con el cursor arriba/abajo y return. Luego debemos responder si vamos a ajustar el tamaño de la ventana, con la dimensión de la pantalla elejida. En caso negativo pregunta si cargamos en la posición del cursor y procede.